**Digit Recognizer**

Zach Brazil, Richard Charles, Adam Kiehl, Zane

**Introduction:**

The Kaggle *Digit Recognizer* prediction competition challenges its participants to classify handwritten numeric digits using computer vision techniques. Over 40,000 labeled digit images were supplied as a training data set and the objective of the competition was to achieve the highest prediction accuracy on a test data set of almost 30,000 images. This problem was approached from an advanced machine learning perspective that leveraged simple and convolutional neural networks (CNNs). This decision was made in part due to the flexibility of CNNs and so every group member was able to pursue their own experimental implementation of the general framework. Below, the problem will be discussed in greater detail and the specifics of each group member’s experimentation will be reviewed.

**Problem Description:**

The Digit Recognizer Kaggle project serves to give new deep learners a good opportunity to get hands-on experience with deep learning via TensorFlow and Keras. This project is to be done with the MNIST ("Modified National Institute of Standards and Technology"). This dataset of handwritten digits has become a classic in the data science community. It is even referred to as the “Hello World” dataset of deep learning. The goal of this Kaggle project is to correctly identify digits from a dataset of tens of thousands of handwritten images. This project serves as a very helpful jumping point for diving deeper into deep learning. The application of deep learning that mostly correlates to this project is computer vision. Computer vision is a field of AI that allows computers to derive meaningful information from digital images and other visual inputs. Some applications of computer vision are object tracking, self-driving cars, and biometrics.

**Quality of Solution:**

* Linear Model:
  + To begin the project, a linear model was first used to see how accurate, or inaccurate, a simple model would be at a classification problem. One single layer was used to compile our model while using the SoftMax activation function. SoftMax was used as it works well for multi-class classification problems, and for the instance of a digit recognizer, we have 10 class labels (pixel value between 0 and 9). The loss function, used to measure how our network performed, was sparse categorical cross entropy due to our classes being exclusive from one another. Finally, the metric used was accuracy as the accuracy of each model is the primary focus of this project. The model was first trained using the training data provided by Kaggle, then tested on the testing data, again provided by Kaggle. The maximum validation accuracy was 92.4% and 90.2% accuracy score on Kaggle. The accuracy was very solid for such a simple model, but we will see future models that are a bit more complex have greater success in a classification setting.
* 2-Layer Fully Connected Model:
  + The second model used was a 2-layer neural network. So, in addition to the one output layer from the linear model, we know have one hidden layer to add a bit more complexity to our solution. Like the output layer from the linear model, the added dense layer will have an activation function, but its function will be ReLu (rectified linear activation function). We don’t use SoftMax as that is a function for classifying the output of our model and since this additional layer isn’t our output layer, SoftMax is avoided. ReLu is a common activation function for dense layers for many types of neural networks due to the ease at which it can train models and the performance is typically satisfactory. Just like the model before, we train our current model using the training data and then test on the testing data, both provided by Kaggle. The maximum validation accuracy was 98.77% and 96.4% accuracy score on Kaggle.
* 9-Layer Fully Connected Model:
  + The solution was done with the Keras package from TensorFlow. The model that was generated was a sequential model with 9 layers. There was a sequence of Conv2D, MaxPooling2D, Flatten, and Dense layers that added enough convolutions to make a more accurate model. When running the model with 25 epochs, the accuracy of the model comes out to 98.74%. This is a value that is quite similar to the other submissions of the project.
* 2D-Convolution Layers
  + This model utilizes depth wise 2d convolution layers to classify the written digits. Rather than using a convolution kernel to produce tensors of outputs with its input channels, these layers use a depth wise kernel to split and convolve its input channels to output to the rest of the layers in the model. One parameter used for each layer, "depth\_multiplier", determines how many output channels are generated for every input channel. Through the time working with these layers, this parameter impacted the model's overall accuracy the greatest as increasing it appeared to increase the model's accuracy, but at the cost of more complex epochs for the model thus increasing runtime when training and testing the model. In comparison to the standard 2d convolution layer, depth wise 2d convolution layers appear to be as effective in terms of accuracy but at the cost of extra runtime. As such, it's hard to see the benefits of using these layers over the former. Though, these layers could perhaps be more effective in a different classification problem.
* Convolutional Neural Network:
  + The CNN model we used was straightforward in design but used the concept of max pooling to highlight the lines of the handwritten digits. The input layer was shaped to receive a 28 by 28-pixel image an­d used the rectified linear unit (ReLU) activation function, a common choice for computer vision models. Another hidden convolutional layer using the same activation function was also added to the model. Following each convolutional layer, a max pooling layer was applied with a pool size of three. This highlights the most prominent value in each group of values it processes, abstracting each layer’s output. The results were then flattened and passed into a dense output layer with ten output values. The SoftMax activation function was chosen for its properties stated above. Of the ten output values, the largest one was selected as the classification prediction. The model achieved a maximum validation accuracy of 98.70%. When the predicted labels for the almost 30,000 test images were submitted to Kaggle, they received an accuracy score of 98.77%
* Final Model:
  + In our final model, we combined bits and pieces from each of the models described above into one and achieved an accuracy rate of 98.5% on Kaggle. With two convolution layers, a depth wise convolution layer, and added dense layers, the model was able to predict with great accuracy without overfitting. This final model didn’t provide the best results out of the several models that were attempted on this dataset. Other than the combination of different layers, everything other aspect of the model remained the same, such as each of the hidden layers had an activation function, ReLu, while the output layer had an activation function of SoftMax just like previous models.

**References**

“Introduction to Virtual Reality: Deep Learning with Tensorflow: Udacity Free Courses.” *Udacity*, https://www.udacity.com/course/intro-to-tensorflow-for-deep-learning--ud187.

“Learn Intro to Deep Learning Tutorials.” *Kaggle*, <https://www.kaggle.com/learn/intro-to-deep-learning>.

“The Mnist Database.” *MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges*, http://yann.lecun.com/exdb/mnist/index.html.

“Module: TF  :   Tensorflow Core v2.8.0.” *TensorFlow*, https://www.tensorflow.org/api\_docs/python/tf.

“What Is Computer Vision?” *IBM*, https://www.ibm.com/topics/computer-vision.